KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode – 638 060

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

AUTOMATIC TEMPERATURE - CONTROLLED FAN USING AT89C51 MICROCONTROLLER

A MICRO PROJECT REPORT

For

22ITC41 - Programming in Python Laboratory

Submitted by

KAVINKUMAR K (23ECR100)

KISHOREKUMAR S (23ECR117)

LOGESHWAR B (23ECR120)

IoT Device Simulator + Dashboard Monitoring System

1. Introduction

In the age of connected devices, real-time monitoring and automation are critical. This project implements a simple yet effective IoT solution to monitor temperature and humidity using an ESP32 and DHT11 sensor. The sensor data is transmitted to a Flask-based server over WiFi and visualized on a live dashboard using Chart.js.

2. Objectives

- Read and monitor temperature and humidity in real-time.

- Transmit sensor data over a wireless network.

- Display data on a responsive and interactive web dashboard.

- Provide regular updates and live charts to users.

- Use lightweight technologies suitable for edge IoT devices.

3. Components Required

ESP32 Board – 1

DHT11 Sensor – 1

Jumper Wires – As required

Breadboard – 1

Power Supply – 1

PC/Laptop – 1

4. System Architecture

DHT11 Sensor → ESP32 Microcontroller → Flask Web Server → Web Dashboard with Live Graphs

5. Circuit Design

Connections:

DHT11 VCC → ESP32 3.3V

DHT11 GND → ESP32 GND

DHT11 Data → ESP32 GPIO 4

6. Software Implementation

ESP32 reads data and posts JSON to a Flask server. Flask stores the latest values and serves them to the frontend. The frontend uses Chart.js to display live data updates.

7. Methodology

1. Connect the DHT11 to ESP32.

2. Program the ESP32 using Arduino IDE.

3. Run the Flask server.

4. Open the web dashboard.

5. Observe live data updates.

8. Use Cases

- Smart Homes

- Agriculture Monitoring

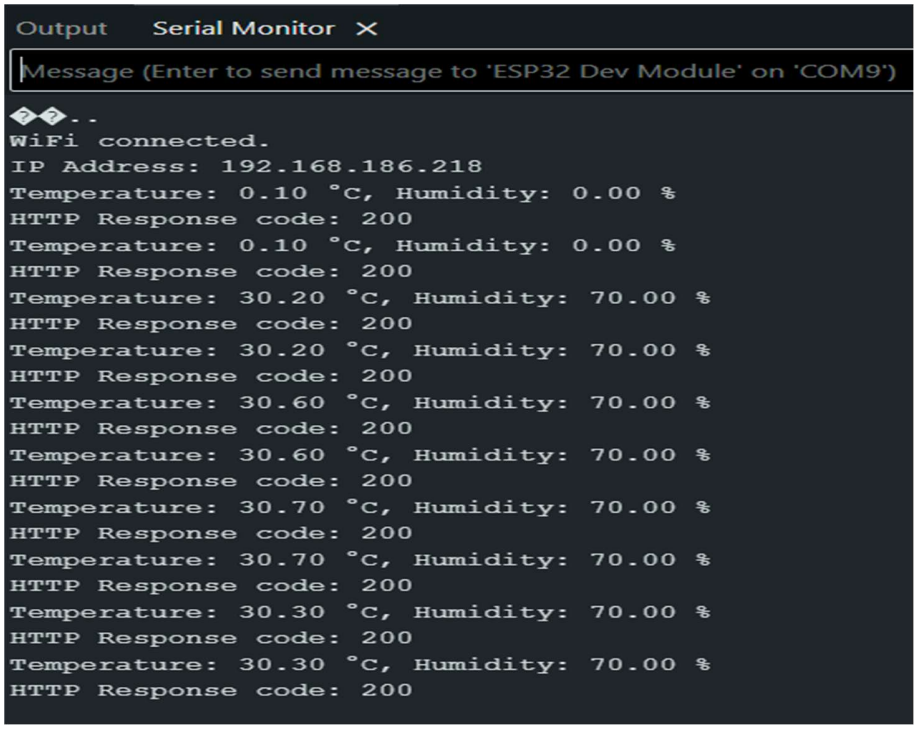
- Server Rooms

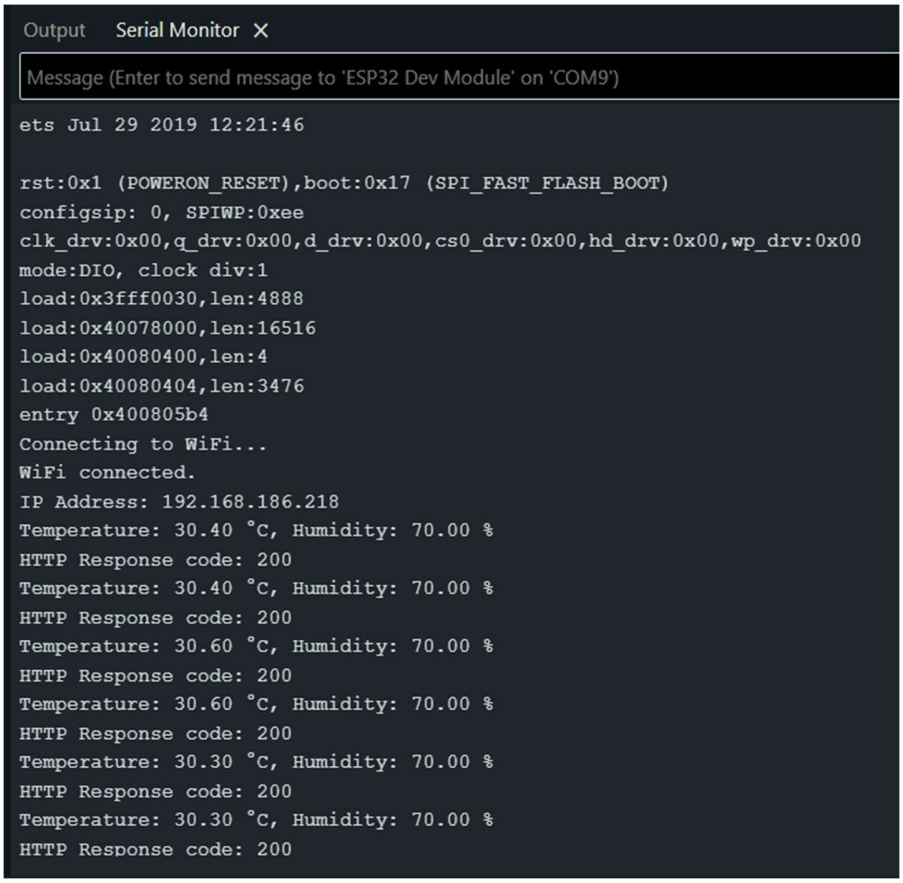
- Industrial Automation

- Healthcare Environments

9. Results and Output

The system displays temperature and humidity in real-time using a responsive web dashboard. Live charts update every 1 seconds to reflect new sensor readings.

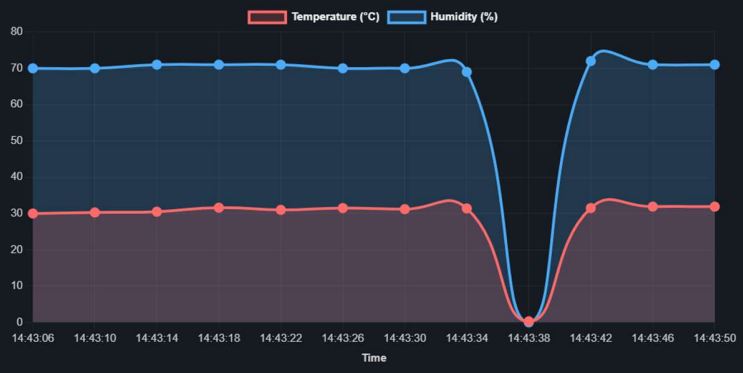





```
PS K:\Micro Project\4th Sem Micro Project\PYTHON> python app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.186.102:5000
Press CTRL+C to quit
127.0.0.1 - - [25/May/2025 14:39:08] "GET /live-data HTTP/1.1" 200 -
Received: {'temperature': 30.2, 'humidity': 70.0, 'timestamp': '2025-05-25 14:39:09'}
192.168.186.218 - - [25/May/2025 14:39:09] "POST /data HTTP/1.1" 200 -
127.0.0.1 - - [25/May/2025 14:39:10] "GET /live-data HTTP/1.1" 200 -
Received: {'temperature': 30.2, 'humidity': 70.0, 'timestamp': '2025-05-25 14:39:10'}
192.168.186.218 - - [25/May/2025 14:39:10] "POST /data HTTP/1.1" 200 -
Received: {'temperature': 30.5, 'humidity': 70.0, 'timestamp': '2025-05-25 14:39:12'}
192.168.186.218 - - [25/May/2025 14:39:12] "POST /data HTTP/1.1" 200 -
Received: {'temperature': 30.5, 'humidity': 70.0, 'timestamp': '2025-05-25 14:39:13'}
192.168.186.218 - - [25/May/2025 14:39:13] "POST /data HTTP/1.1" 200 -
127.0.0.1 - - [25/May/2025 14:39:14] "GET /live-data HTTP/1.1" 200 -
Received: {'temperature': 30.1, 'humidity': 70.0, 'timestamp': '2025-05-25 14:39:14'}
192.168.186.218 - - [25/May/2025 14:39:14] "POST /data HTTP/1.1" 200 -
Received: {'temperature': 30.1, 'humidity': 70.0, 'timestamp': '2025-05-25 14:39:15'}
192.168.186.218 - - [25/May/2025 14:39:15] "POST /data HTTP/1.1" 200 -
Received: {'temperature': 30.3, 'humidity': 70.0, 'timestamp': '2025-05-25 14:39:17'}
192.168.186.218 - - [25/May/2025 14:39:17] "POST /data HTTP/1.1" 200 -
Received: {'temperature': 30.3, 'humidity': 70.0, 'timestamp': '2025-05-25 14:39:18'}
192.168.186.218 - - [25/May/2025 14:39:18] "POST /data HTTP/1.1" 200 -
```

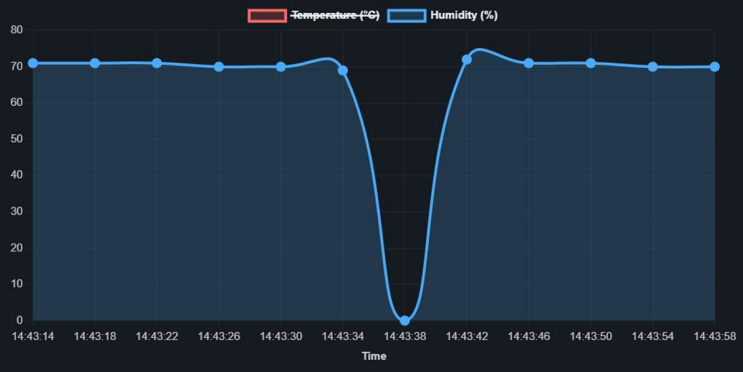
Live IoT Temperature & Humidity Dashboard

Temp: 31.9 °C | Hum: 71.0 %



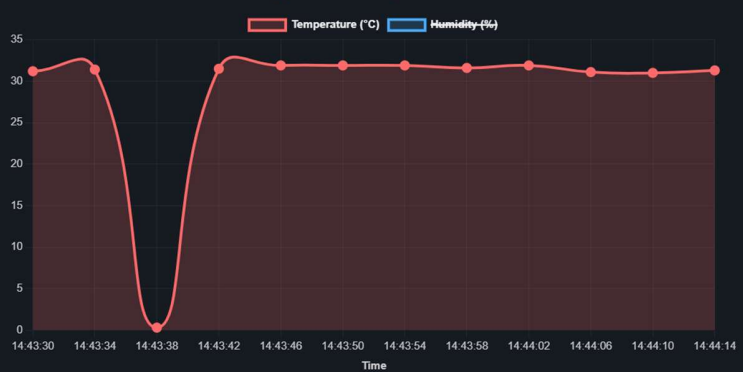
Live IoT Temperature & Humidity Dashboard

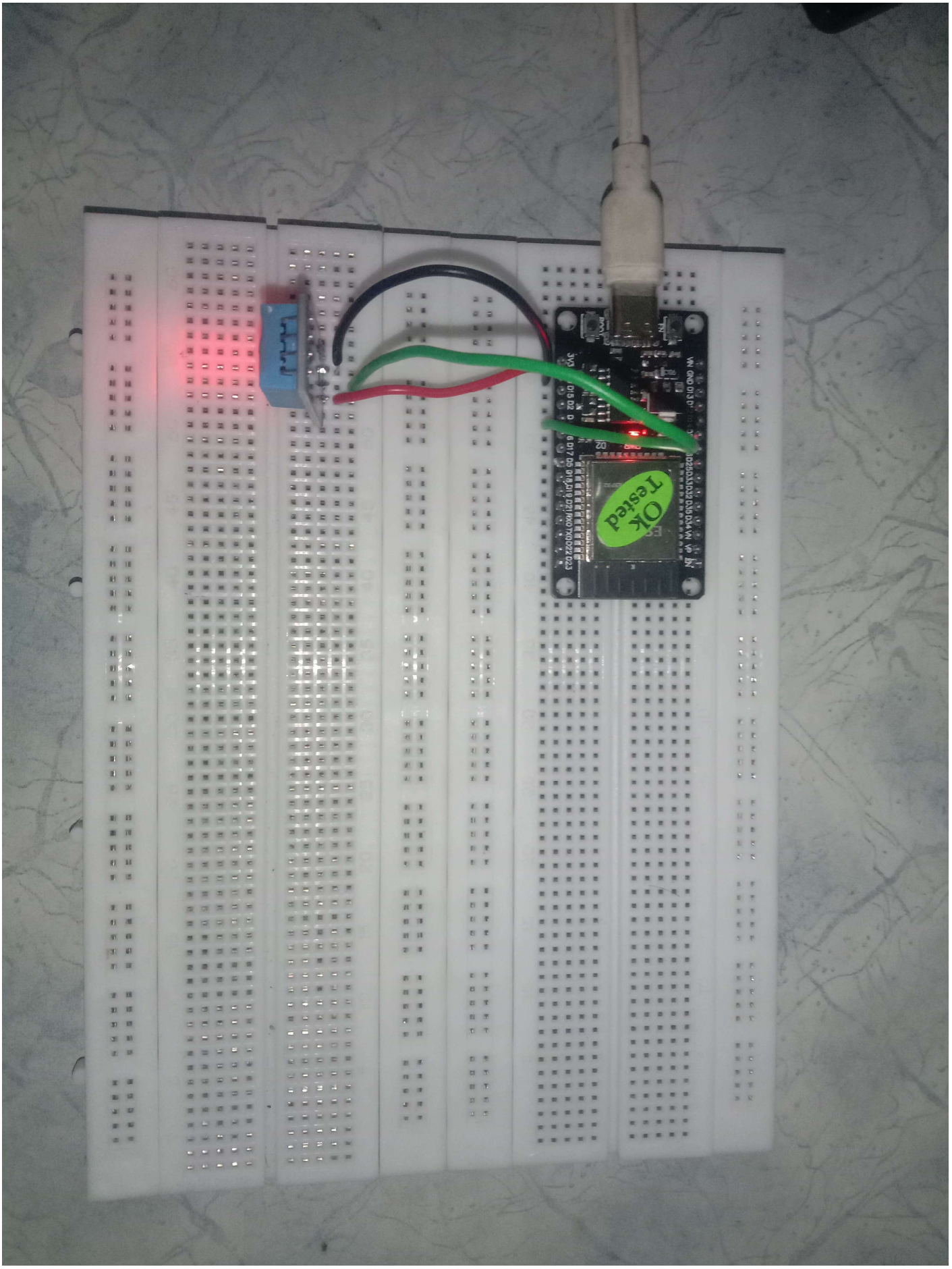
Temp: 31.6 °C | Hum: 70.0 %



Live IoT Temperature & Humidity Dashboard

Temp: 31.3 °C | Hum: 69.0 %




Ok
Tested

10. Advantages and Limitations

Advantages:

- Real-time data

- Wireless system

- Open-source

Limitations:

- Limited sensor accuracy

- Local network only

11. Future Enhancements

- Add database storage

- Use DHT22 or BME280 sensors

- Add alerts for thresholds

- Host on cloud

- Make dashboard mobile responsive

12. Appendix

ESP32 Arduino Code :

```
#include <WiFi.h>

#include <HTTPClient.h>

#include "DHT.h"


#define DHTPIN 4

#define DHTTYPE DHT11


DHT dht(DHTPIN, DHTTYPE);


const char* ssid = "Kishore63k";
```

```
const char* password = "6374864213";

const char* serverURL = "http://192.168.186.102:5000/data";

void setup() {
  Serial.begin(115200);          // Make sure Serial Monitor baud = 115200
  dht.begin();

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected.");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Check if readings are valid
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
  } else {
```

```
  // Print to Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C, Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");

  // Prepare JSON payload
  HTTPClient http;
  http.begin(serverURL);
  http.addHeader("Content-Type", "application/json");

  String payload = "{\"temperature\":" + String(temperature) + ",\"humidity\":" + String(humidity) + "}";
  int httpResponseCode = http.POST(payload);

  Serial.print("HTTP Response code: ");
  Serial.println(httpResponseCode);

  http.end();
 }

 delay(1000);  // Wait 1 seconds before next reading
}
```

Flask Backend Code :

```
from flask import Flask, request, jsonify, render_template
```

```
import time


app = Flask(__name__)
latest_data = {"temperature": 0, "humidity": 0, "timestamp": ""}


@app.route('/')
def dashboard():
    return render_template('dashboard.html', data=latest_data)


@app.route('/data', methods=['POST'])
def receive_data():
    global latest_data
    content = request.json
    latest_data["temperature"] = content["temperature"]
    latest_data["humidity"] = content["humidity"]
    latest_data["timestamp"] = time.strftime("%Y-%m-%d %H:%M:%S")
    print("Received:", latest_data)
    return jsonify({"status": "success"})


@app.route('/live-data', methods=['GET'])
def live_data():
    return jsonify(latest_data)


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

HTML + JS Dashboard Code :

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1" />
 <title>Modern IoT Dashboard</title>
 <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
 <style>
  body {
   background: #0d1117;
   color: #fff;
   font-family: 'Segoe UI', sans-serif;
   text-align: center;
   padding: 30px;
  }

  h1 {
   font-size: 2em;
   margin-bottom: 10px;
   color: #58a6ff;
  }

  .values {
   font-size: 1.5em;
   margin: 15px;
  }

  .card {
```

```
    background: #161b22;
    padding: 20px;
    border-radius: 16px;
    box-shadow: 0 0 25px rgba(0, 0, 0, 0.5);
    max-width: 800px;
    margin: auto;
  }

  canvas {
    max-width: 100%;
  }
 </style>
</head>
<body>
 <h1>Live IoT Temperature & Humidity Dashboard</h1>

 <div class="card">
  <div class="values">
   🌡️ Temp: <span id="temp">--</span> °C | 💧 Hum: <span id="hum">--</span> %
  </div>
  <canvas id="iotChart"></canvas>
 </div>

 <script>
  const ctx = document.getElementById("iotChart").getContext("2d");

  const iotChart = new Chart(ctx, {
```

```
type: "line",
data: {
  labels: [],
  datasets: [
    {
      label: "Temperature (°C)",
      data: [],
      borderColor: "#ff6b6b",
      backgroundColor: "rgba(255, 107, 107, 0.2)",
      fill: true,
      tension: 0.5,
      pointRadius: 5,
      pointHoverRadius: 8,
      pointBackgroundColor: "#ff6b6b",
    },
    {
      label: "Humidity (%)",
      data: [],
      borderColor: "#4dabf7",
      backgroundColor: "rgba(77, 171, 247, 0.2)",
      fill: true,
      tension: 0.5,
      pointRadius: 5,
      pointHoverRadius: 8,
      pointBackgroundColor: "#4dabf7",
    },
  ],
```

```
},
options: {
 responsive: true,
 animation: {
  duration: 1000,
  easing: "easeOutQuart"
 },
 scales: {
  x: {
   ticks: { color: "#ccc" },
   grid: { color: "rgba(255,255,255,0.05)" },
   title: {
    display: true,
    text: "Time",
    color: "#ccc",
    font: { weight: "bold" }
   }
  },
  y: {
   ticks: { color: "#ccc" },
   grid: { color: "rgba(255,255,255,0.05)" },
   beginAtZero: true
  }
 },
 plugins: {
  legend: {
   labels: {
```

```
        color: "#eee",
        font: { weight: "bold" }
       }
      },
      tooltip: {
       backgroundColor: "rgba(0,0,0,0.8)",
       borderColor: "#444",
       borderWidth: 1
      }
     }
    }
});

async function fetchData() {
  try {
    const response = await fetch("/live-data");
    const result = await response.json();
    const now = new Date().toLocaleTimeString();

    document.getElementById("temp").textContent = result.temperature.toFixed(1);
    document.getElementById("hum").textContent = result.humidity.toFixed(1);

    if (iotChart.data.labels.length >= 12) {
      iotChart.data.labels.shift();
      iotChart.data.datasets[0].data.shift();
      iotChart.data.datasets[1].data.shift();
    }
```

```
      iotChart.data.labels.push(now);
      iotChart.data.datasets[0].data.push(result.temperature);
      iotChart.data.datasets[1].data.push(result.humidity);
      iotChart.update();

    } catch (err) {
      console.error("Fetch error:", err);
    }
  }

  setInterval(fetchData, 4000);
  fetchData();
 </script>
</body>
</html>
```

12. Conclusion

This project demonstrates how IoT can monitor environmental conditions in real time. It provides a foundation for more advanced and scalable systems.